Aim: To understand the working principle of Artificial Neural network with feed forward and feed backward principle.

Program: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

BACKPROPAGATION Algorithm

BACKPROPAGATION (*training_example*, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form $(\vec{x}, \vec{t})$*, where* $(\vec{x})$ *is the vector of network input values,* $(\vec{t})$ *and is the vector of target network output values.*
η *is the learning rate (e.g., .05).* n_i*, is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*
The input from unit i into unit j is denoted x_{ji}*, and the weight from unit i to unit j is denoted* w_{ji}

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do

    - For each $(\vec{x}, \vec{t})$, in training examples, Do

      Propagate the input forward through the network:
      1. Input the instance $\vec{x}$, to the network and compute the output o_u of every unit u in the network.

      Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = \eta \delta_j x_{i,j}$$

Training Examples:

Example	Sleep	Study	Expected % in Exams
1	2	9	92
2	1	5	86
3	3	6	89

Normalize the input

Example	Sleep	Study	Expected % in Exams
1	2/3 = 0.66666667	9/9 = 1	0.92
2	1/3 = 0.33333333	5/9 = 0.55555556	0.86
3	3/3 = 1	6/9 = 0.66666667	0.89

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100

# Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))
```

```python
# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

# Variable initialization
epoch=500 # Setting training iterations
lr=0.1 # Setting learning rate
inputlayer_neurons = 2 # number of features in data set
hiddenlayer_neurons = 3 # number of hidden layers neurons
output_neurons = 1 # number of neurons at output layer

# weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

# draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    # Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)

    #Backpropagation
```

```
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)

    #how much hidden layer wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

    # dot product of next layer error and current layer output
    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr

print("\n Input: \n" + str(X))
print("\n Actual Output: \n" + str(y))
print("\n Predicted Output: \n" ,output)
```

Input:

```
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
```

Actual Output:

```
[[0.92]
 [0.86]
 [0.89]]
```

Predicted Output:

```
 [[0.89511671]
 [0.8828429 ]
```

[0.89243792]]